Web-based application for the Program Algebra Toolset

Pim Duin

August 14, 2009

Supervisor(s): Inge Bethe (UvA), Bob Diertens(UvA)
Signed:
Abstract

Program algebra (PGA) helps the understanding of sequential programming and its concepts are a great introduction to computer science. Program algebra is used at a web based course for students with pre-university education to get familiar with both computer science and the university. In support of program algebra a toolset has been developed. This thesis is a description of the development of a web based frontend to make the program algebra toolset more accessible to students who follow the web based course.
TABLE OF CONTENTS

1. Introduction ......................................................................................................................................... 7
   1.1 Context ......................................................................................................................................... 7
   1.2 Goal ........................................................................................................................................... 7
   1.3 Research procedure ......................................................................................................................... 7
2. Program algebra .................................................................................................................................. 9
   2.1 PGA ........................................................................................................................................... 9
   2.2 Program notations, projections and embedding ............................................................................. 9
3. Current toolset .................................................................................................................................... 11
4. Comparison of Languages/Techniques ........................................................................................... 13
   4.1 Criteria ....................................................................................................................................... 15
   4.2 Contestants .................................................................................................................................. 15
      4.2.1 Java Applet + Remote method invocation ............................................................................. 16
      4.2.2 Ruby on Rails ......................................................................................................................... 17
      4.2.3 Google Web Toolkit ............................................................................................................... 17
   4.3 Comparison and Conclusion .......................................................................................................... 18
      4.3.1 Accessibility .......................................................................................................................... 18
      4.3.2 Support ................................................................................................................................. 18
      4.3.3 Responsiveness ...................................................................................................................... 19
      4.3.4 Performance ........................................................................................................................ 19
      4.3.5 Usability .................................................................................................................................. 19
      4.3.6 Maintainability ....................................................................................................................... 19
      4.3.7 Expandability ......................................................................................................................... 19
      4.3.8 Applicability ........................................................................................................................ 20
   4.4 Final choice ................................................................................................................................... 20
5. Designing the Application .................................................................................................................. 21
   5.1 Examining the functional requirements .......................................................................................... 21
   5.2 Identifying the elements of the UI design ....................................................................................... 22
   5.3 Views ........................................................................................................................................... 22
      5.3.1 Program Bar ............................................................................................................................ 22
      5.3.2 Simulator view ......................................................................................................................... 23
      5.3.3 Editor view ............................................................................................................................. 24
      5.3.4 Projections view .................................................................................................................... 25
6. Building the User Interface ................................................................................................................. 27
   6.1 Selecting GWT widgets to implement the UI elements ................................................................. 27
   6.2 Selecting GWT panels to layout the UI elements .......................................................................... 27
7. Back-end ............................................................................................................................................. 31
7.1 Google Web Toolkit remote procedure call ................................................................. 31
7.2 Remote procedure calls .............................................................................................. 32
    7.2.1 Stateless versus stateful ....................................................................................... 32
    7.2.2 Program Bar ......................................................................................................... 32
    7.2.3 Projection view .................................................................................................... 33
    7.2.4 Simulator view .................................................................................................... 33
8. Conclusion ....................................................................................................................... 35
9. References ....................................................................................................................... 37
1. INTRODUCTION

1.1 CONTEXT

Every year a web class is organized. Its goal is to make students with pre-university education to get familiar with both the study of computer science and the university. The web class is, as its name suggests, web based. Students who apply for the course only use the course’s website\(^1\) to learn about program algebra (PGA). Program algebra is used in the course to give the students an idea about what a program is. The web class website contains some study material and exercises about program algebra.

1.2 GOAL

In support of the understanding and research of program algebra a toolset\(^2\) has been written. The toolset is designed to work on UNIX machines which could make it hard for web class students to get it to work on their own machines. It would be nice if the toolset can be used by the web class students to help them understand program algebra.

To make the toolset more accessible a web based frontend to a part of the toolset is created. This web based application includes simulation of certain programs and can be run using the Firefox web browser. It will also run on different browsers but it will not always work that well because most browsers keep their own standards.

The main goal is (literally) to build platform independent web-based applications for the PGA-Toolset for the support of the Webklas Informatica: Wat is een programma?

1.3 RESEARCH PROCEDURE

It is clear that the focus of this project is on the engineering side. However there is also some research to do. The first thing is to make a choice about how the application is implemented; which language and techniques should be used. After this choice has been made, the design of the application has to be made, both frontend (the user interface) and backend (remote procedure calls).

---

\(^1\) [http://www.science.uva.nl/research/prog/projects/pga/](http://www.science.uva.nl/research/prog/projects/pga/)

2. PROGRAM ALGEBRA

Because this thesis uses a few concepts from program algebra [1] it is imperative to dedicate a chapter to the basics of it.

2.1 PGA

Program algebra is an algebraic framework of sequential programming. It is introduced almost a decade ago to help the understanding of sequential programming by developing the most simple and basic concept of a programming language. The algebra works with several constants and operators. The constants can be thought of as instructions. Program algebra consists of the following instructions:

- **Basic instruction:** a
  All elements \( a \in \Sigma \) are basic instructions. After performing an instruction the next instruction is executed.

- **Termination instruction:** !
  Upon execution of this instruction the further execution of a program stops.

- **Positive test instruction:** +a
  If this instruction returns the boolean true the next instruction will be executed. If it returns false the next instruction will be skipped.

- **Negative test instruction:** -a
  If this instruction returns the boolean false the next instruction will be executed. If it returns true the next instruction will be skipped.

- **Forward jump instruction:** #k
  This and the next \( k - 1 \) instructions are skipped.

The operators which can tie these instructions together are:

- **Concatenation**
  Concatenation of instructions a and b. Notation: \( a;b \)

- **Repetition**
  Repetition of instruction a. Notation \( a^\omega \)

2.2 PROGRAM NOTATIONS, PROJECTIONS AND EMBEDDING

PGLA is a program notation for representing PGA expressions. It adds another kind of instruction to PGA which replaces repetition. On top of PGLA other program notations are designed which map onto PGLA. These program notations can add a higher level of abstraction by adding instructions. These other program notations can be mapped on each other too. These mappings from one program notation to another are either called projections or embeddings. Figure 1 shows the hierarchy of program notations developed.

---

\(^3\) \( \Sigma \) is the set of basic instructions.
A transition from a higher level (in figure 1 literally higher in the figure) to a lower level program notation is called a projection. Transitions from lower level program notation to a higher level are called embeddings.
3. CURRENT TOOLSET

To support research and education about program algebra a toolset has been written [2]. The tools are written using Perl [3], with the exception of the bisimulation tester. The bisimulation tester is implemented using the C [4] language for performance reasons. For the graphical views Tcl/Tk [5] is used. The toolset is written to be used on UNIX machines. With a little work it can also be used on other operating systems. Because the toolset is used for research, it is written to be easily extendible. The toolset consists (among others) of:

- **Parser**
  
The parser checks the syntax of programs. It accepts several programming languages and basic instructions.

- **Projections**
  
  Several projections have been implemented which can be used to map one program notation to another.

- **Simulator**
  
The simulator accepts several programming languages and basic instruction sets. It is used to explore a program step-by-step. It has a textual interface and an optional graphical interface. Also, if available, a graphical view of the core of the basic instruction can be given.

- **Bisimulation tester**
  
The bisimulation tester tests for bisimularity. In other words: it checks if two programs behave in the same way. Because tests for bisimularity can be quite computationally expensive the bisimulation tester is implemented using the C language.

The simulator has an online graphical implementation to aid the web class students (see figure 2). It is implemented using HTML\(^4\), CSS\(^5\), Perl and PHP [6] and makes use of remote procedure calls to the toolset; the users use the command line toolset through these remote procedure calls.

\(^{4}\) http://www.w3.org/html/

\(^{5}\) http://www.w3.org/css/
Figure 2: Screenshot of the online graphical simulator application
4. COMPARISON OF LANGUAGES/TECHNIQUES

There are several ways to make the toolset available through a web browser. The first method is to port the required tools to languages available to web browsers. Perl and Tcl/Tk code (with which the majority of the toolset has been written) cannot be run in a web browser so every bit of code has to be ported to languages which are supported by web browsers. This would not be a problem if the toolset was not that big in size. However, the toolset is currently implemented using thousands of lines of code.

So porting the toolset is not a good idea. Not only because it will be a lot of work (well beyond the scope of this project) but also because the toolset already is available. A better way is to follow the path that the PHP implementation of the online toolset was already starting; make the application to an interface between the user and the toolset. This way the already present toolset is used and with a relatively small amount of code the toolset is made available to the user in an easy way.

Obviously, using the application to be a mediator between user and actual toolset is the choice I made to implement the project. The next choice will be the choice of languages and techniques used for the actual implementation.

There are many different ways to implement the web based application. The number of languages and techniques available to implement web based applications is staggering. Languages and techniques available for the client and server will have to be considered. In order to make this a little more clear I made a little model (Figure 3).

![Client/server model](image-url)

*Figure 3: Client/server model*
Every numbered element in figure 3 has to be filled with one or more languages of techniques. The arrows stand for communication.

1. Web client
   At the client side of the application only languages and techniques supported by web browsers can be used. This may sound like it really limits the possibilities but there are quite a few choices.
   - HTML/CSS
   - Javascript [7]
   - Java Applet [8]
   - Flash (actionscript)\(^6\)
   - Browser plugin based (ActiveX, Firefox XUL\(^8\))

   Of course, the web browser is not tied to only one of these choices, most of these languages/techniques can be used together.

2. Server
   On the server side the only requirement of the languages and techniques used is that it has to be compatible with the client side languages. In other words: there have to be languages and techniques available to glue the client and server side together (fill in the arrows between client and server in figure 3).

3. Toolkit
   The toolkit has already been written.

4. Client to server communication
   There are a few ways to communicate from client to server. This can be done by, for example, HTTP\(^9\) GET and POST requests, JavaScript Object Notation\(^{10}\) or XML\(^{11}\).

5. Server to client communication
   This type of communications can be done using JavaScript Object Notation or XML for example.

6. Toolkit to server communication
   The command-line output of the toolkit is passed on to the server.

7. Server to toolkit communication
   This will simply just be the server invoking the toolkit with the correct parameters.

Numbers three, six and seven of figure 3 are not variable and do not have to be covered by any language or technique.

In the next chapter I will pick some languages and techniques, place them inside the model and compare them.

\(^{6}\) http://www.adobe.com/devnet/flash/
\(^{7}\) http://en.wikipedia.org/wiki/ActiveX
\(^{8}\) https://developer.mozilla.org/En/XUL
\(^{9}\) http://www.w3.org/Protocols/
\(^{10}\) http://www.json.org/
\(^{11}\) http://www.w3.org/XML/
4.1 CRITERIA

As mentioned in the last paragraph, there are a lot of languages and techniques to choose from. Almost any combination of languages can be used to implement the application. There is a lot to choose from, but the choice has to be made. It would be impossible to thoroughly investigate every possibility so I will choose a few, compare them and make a decision.

In order to be able to compare any languages or techniques I will first need a set of criteria to compare the languages and techniques by. There are dozens of criteria that can be used for comparing. The following eight are the most important:

- **Accessibility**
  The application should be accessible; it should be able to be reached by as much users as possible. The application should support most of the popular web-browsers (Internet Explorer, Firefox, Mozilla, Safari and Opera). This is a very important criterion which follows almost directly from the project’s goal; to make the toolset more accessible to the web class students.

- **Support**
  The languages and techniques should be well supported. It would be nice if the languages and techniques are actively developed. Good manuals, tutorials or user forums can be very helpful.

- **Responsiveness**
  The languages used should make the program to be responsive to user input. The languages/techniques should make the application respond quickly to the user’s actions.

- **Performance**
  The languages used should be able to produce efficiently executable code. Network connections should also be used efficiently. Client side web based applications generally are slower than their desktop counterparts but they should not ‘feel’ slow. The network should not be unnecessarily burdened.

- **Usability**
  The languages used should be able to create an intuitive user interface, and this interface should be easy to use. Of course, creating a nice user interface will mostly be the programmer’s work, but the language can help by, for example, providing tools which can help the programmer build a user interface.

- **Maintainability**
  Maintenance on the applications should be a short as possible. Again, maintainability of an application depends heavily on the implementation. Maintainability can be increased if, for example, debuggers are present.

- **Expandability**
  New features should be easy to add. Tools can help, but again, expandability will depend mostly on the implantation.

- **Applicability**
  Here I address the applicability of the technique; the implementation should be quick and easy. It would be nice if, for example, the technique abstracts low level code.

4.2 CONTESTANTS

I made a selection of a few languages and techniques to use. In the next paragraph I will explain why I chose these and which characteristics they have. I also describe how they fit the criteria mentioned above.
It would be impossible to investigate every possibility but fortunately nowadays there are quite a few web toolkits which incorporate different languages and/or techniques. The toolkits save time by, for example, abstracting networking code, providing easy debugging or shipping useful libraries. Because these toolkits are specifically designed to make creating a web application easier I chose two of the contestants to be toolkits. The last contestant is a more traditional approach using Java. First I will judge the traditional approach using the criteria.

### 4.2.1 JAVA APPLET + REMOTE METHOD INVOCATION

I chose the Java applet as a contestant because it is a mature and trusted technique; Java applets have been around for many years. This method will use a Java applet on the client side and Java code on the server side. It will use remote method invocation\(^\text{12}\) for communication between the web client and the server. Remote method invocation is the Java implementation of a remote procedure call. It allows an object to invoke a method on another object running in a different Java virtual machine.

- **Accessibility**
  The applet uses Java, so it works with all browsers which support Java applets.

- **Support**
  Java has been around for nearly two decades and is a popular and widely used language. It is actively developed and it is safe to say it will be supported for years to come.

- **Responsiveness**
  The responsiveness of the applet will be OK. Asynchronous communication can be implemented with some work\(^\text{13}\).

- **Performance**
  Although improved over the years Java's performance lags a bit behind on other languages. However, for this project it will be performing sufficiently. A slight disadvantage is that booting the Java runtime environment takes a few seconds.

- **Usability**
  Creating a user interface using Java is easy. The Java library features all of the popular user interface elements like panels, buttons and text fields. An integrated development environment called Netbeans\(^\text{14}\) even allows the developer to create a user interface by dragging and dropping user interface elements onto a window.

- **Maintainability**
  There are a few integrated development environments supporting the Java like Eclipse\(^\text{15}\) or the aforementioned Netbeans. There are also visual debuggers available.

- **Expandability**
  The object oriented language helps a bit but expandability fully depends on the implementation of the application. Also, using Java on the client side has the advantage to easily bring toolset functionality to the client side, lowering server load.

- **Applicability**
  An applet is easy to get running. Java's remote method invocation makes remote procedure calls easy. User interface programming is easy because of the available user interface elements. However, in order to make the application responsive the developer has to implement asynchronous communication himself.

---

\(^{12}\) [http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp](http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp)


\(^{14}\) [http://www.netbeans.org/](http://www.netbeans.org/)

\(^{15}\) [http://www.eclipse.org/](http://www.eclipse.org/)
4.2.2 RUBY ON RAILS

I chose Ruby on Rails\(^{16}\) as a contestant by recommendation. It is supposed to be the new big thing in web development and it is getting more popular by the minute. After looking into Ruby on Rails I thought it would be fitted to be used as method of implementation.

Ruby on Rails is a web development platform based on the Ruby language. It is based on the model-view-controller model\(^{17}\) and it includes different tools to make web development easier. Rails provides tools which can automatically generate some code for basic websites. The majority of these tools are focused on database code generation. A simple web server and a make\(^{18}\)-like build system are included.

- **Accessibility**
  Ruby on Rails comes with a few JavaScript libraries. These libraries provide cross browser compatibility and support all major browsers.
- **Support**
  Ruby on Rails is up and coming. There are tens of thousands of websites using Ruby on Rails. Ruby on Rails is actively developed\(^{19}\).
- **Responsiveness**
  Using the aforementioned JavaScript libraries a user interface can be built with ease. These user interfaces behave just like expected from a desktop application. With some effort asynchronous remote procedure calls can be made.
- **Performance**
  At time of writing Rails does not work with the newest versions of Ruby yet. The older versions of Ruby are slow compared to the newer versions and other programming languages. Fortunately the Ruby code will not be that big and will not be complex; it is only run on the server and manages communication between the client and the toolkit.
- **Usability**
  The JavaScript libraries can be used to build a user interface.
- **Maintainability**
  Because of the Model-View-Controller model and Rails' favoring of convention over configuration Rails is forcing the developer to work in a specific way. Rails make design decisions for the developer, making it easier for other developers to get into the code. Currently, only command line debuggers are available.
- **Expandability**
  Because of the Model-View-Controller model expandability gets a bit easier.
- **Applicability**
  The Model-View –Controller model is a nice way to build an application on. The JavaScript libraries make user interface programming easy. However, I found the Ruby and Ruby on Rails learning curve quite steep.

4.2.3 GOOGLE WEB TOOLKIT

After a bit of research I discovered the Google Web Toolkit\(^{20}\). This is a fairly new set of tools that allows developers to write JavaScript applications using Java. Its main component is the Java to JavaScript compiler which, as its name suggests, translates Java code to JavaScript code. It also

\(^{16}\) [http://rubyonrails.org/](http://rubyonrails.org/)
\(^{17}\) [http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller](http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller)
\(^{19}\) [http://rubyonrails.org/applications](http://rubyonrails.org/applications)
\(^{20}\) [http://code.google.com/webtoolkit/](http://code.google.com/webtoolkit/)
features a debugging environment and a wide range of ready to use user interface elements. After some investigation I thought it would be a good idea to consider using the Google Web Toolkit.

- **Accessibility**
  All Google Web Toolkit applications automatically support all major browsers.

- **Support**
  The Google Web Toolkit is used by Google internally, and also used by many other companies. During the course of this project Google has updated its toolset a few times. Also the trusted brand ‘Google’ is associated to the toolkit. This all is an indication the Google Web Toolkit will be supported for some time to come.

- **Responsiveness**
  The Google Web Toolkit supports asynchronous remote procedure calls out of the box.

- **Performance**
  Efficiency is among the top priorities of the Google Web Toolkit project. It uses a method called deferred binding. This means that for every browser a single optimized piece of code is compiled. Different browsers run different optimized code but with the same results. This means also bandwidth is saved because browsers do not have to download code they do not use.

- **Usability**
  The Google Web Toolkit ships with an extensive (so-called) widget library. This library contains user interface elements like panels, buttons and text fields. Creating a user interface is just a matter of combining and embedding these widgets.

- **Maintainability**
  The Google Web Toolkit comes with a visual debugger which can be run on both the client and server code. It also features a ‘hosted mode’ browser. In this mode the application is run using only Java code, this allows the programmer to quickly run Google Web Toolkit applications without first having to compile them.

- **Expandability**
  Expandability fully depends on the implementation of the application.

- **Applicability**
  User interface programming is easy because of the user interface elements library. Asynchronous remote procedure calls are already implemented and ready to use. The use of only one language throughout the whole project makes for a nice learning curve.

### 4.3 COMPARISON AND CONCLUSION

Now the characteristics of the languages/techniques are known, it is time to compare them side by side criteria-wise.

#### 4.3.1 ACCESSIBILITY

All of the contestants have the potential to make the application accessible. Most browsers support Java applets and both Google Web Toolkit and Ruby on Rails deliver cross-browser compatibility. All contestants score equal points.

#### 4.3.2 SUPPORT

---

22 [http://gwt.google.com/samples/Showcase/Showcase.html](http://gwt.google.com/samples/Showcase/Showcase.html)
Java has the best support. This mature language has been around for almost two decades and has proven it is here to stay. Ruby on Rails is a lot younger and Google Web Toolkit is the youngest. It is really hard to say whether Ruby on Rails or Google Web Toolkit will have better support. I will put the Google Web Toolkit on the second place because there is a big company behind it. Also, if something is branded 'Google' it generally is considered to be a good product. All contestants don't score equal points here, but they are supported and safe to use.

4.3.3 RESPONSIVENESS

All contestants have the potential to be responsive. They all feature asynchronous communication with the server and the user interfaces are snappy. All contestants score equal points on this criterion.

4.3.4 PERFORMANCE

Because the application will be no more than a front end to the toolset performance will not really be an issue. Both client and server code will not be complex; most computers will have no problem running the client side of the application. Google Web Toolkit's usage of deferred binding will lead to better performance over the JavaScript libraries that Ruby on Rails provides. Both Google Web Toolkit's client code and network usage are more efficient than Ruby on Rails'. It takes a bit longer to load the application as a Java applet than running it using Google Web Toolkit or Ruby on Rails because the Java runtime environment has to be booted. Google Web Toolkit comes in first on the client side, followed by Ruby on Rails and the Java applet respectively. On the server side it is a somewhat different story. Here, because the Ruby language has relatively poor performance both Google Web Toolkit and the Java applet are ranked first. They both use more or less the same Java code on the server side.

4.3.5 USABILITY

All contestants come with a variety of ready to use user interface elements. The Java applet method has a slight advantage over the other techniques because it has a drag and drop user interface creator.

4.3.6 MAINTAINABILITY

Ruby on Rails has its Model-View-Controller model and Google Web Toolkit and the Java applet method have their graphical debuggers. It is not easy to declare a winner in this criterion, but if there has to be a winner it would be the Google Web Toolkit. The hosted mode makes it very easy for the developer to pinpoint bugs on the client side and the server side at the same time. At the other contestants it is a bit harder to find bugs this way. Using Google Web Toolkit only one debugger is needed where the other contestants need debugger on both the client and the server side.

4.3.7 EXPANDABILITY

Expandability increases if the developer sticks to the conventions of Ruby on Rails. The programmer has more control by using the Google Web Toolkit or the Java applet. This means the programmer decides on the expandability of the application. Ruby on Rails wins in criterion because it forces the developer to work in a certain way in which expandability is easy.
4.3.8 APPLICABILITY

All contestants have ready-to-use user interface libraries; so creating a user interface will be easy with all contestants. Google Web Toolkit has an advantage when it comes to asynchronous remote procedure calls; it has been implemented and is easy to use by the developer.

4.4 FINAL CHOICE

The final choice is the Google Web Toolkit. This toolkit has everything that is needed to write the application in an easy way. It features ready to use user interface elements, cross browser compatibility and asynchronous communication out of the box. Where Ruby on Rails is more targeted at web applications using a database Google Web Toolkit is targeted at creating desktop-like web applications. The Java applet method is not particularly targeted at any kind of application. Google Web Toolkits performance is a nice accidental circumstance too. Another factor I considered in the comparison is that I already know Java. I do not have any experience with Ruby and the Rails framework. It’s easier for me to get to know how the Google Web Toolkit works.
5. DESIGNING THE APPLICATION

The design of the application comes in several steps. First I will look at the functional requirements of the application, answering the question: what should the user be able to do with the application? Next, I will give the application an abstract user interface. This means I will map the functionality on a general user interface. In a later chapter I will map actual user interface elements on the general user interface. The actual user interface elements will be implemented.

5.1 EXAMINING THE FUNCTIONAL REQUIREMENTS

In this chapter I will look at the functional requirements of the application. It has to provide at least the functionality of the current toolset in one way or another. First, I list the requirements of each tool of the toolset individually.

Parser

- Load a user program and its properties (language and instruction set).
- Parse the program.
- Indicate if program is correct.

Projections

- Load a user program and its properties (language and instruction set).
- List available projections.
- Apply chosen projections.
- Save the program.

Simulator

- Load a user program
- Can run the program one step at a time.
- Can undo the program one step at a time.
- Can reset the program counter.
- Has a graphical representation of the current state of the program.

Looking at the functional requirements we see that the requirement of loading a program is present in each of the set of requirements of the tools. There are a number of different ways to load a program.

- Load program from the HTTP GET request. This is done by embedding the program in the URL pointing to the web application. The resulting URL can be parsed and the program can be extracted. This is especially handy if students have to work with a pre-written program.
- Load program by submitting a file to the server using HTTP file upload. Uploading contents of a file directly to the client side is prohibited for security reasons. The file first has to be uploaded to the server which sends it back to the client.
- Load program by copy and pasting into a text field. The user copies the program to the clipboard and pasts it into the browser window.

To make the last bullet point above possible I decided to also implement an editor.

Editor
The editor will be very basic; it will basically be a text field in which the user can input a program. It also allows the user to use his or her editor of choice to write programs and use the clipboard to copy and paste them into the editor. It can also be used to edit a program loaded in any other way.

5.2 IDENTIFYING THE ELEMENTS OF THE UI DESIGN

In this paragraph I will explain how I give the tools their user interface. I will map the programs onto the screen using abstract user interface elements (buttons, input fields, etcetera), which in a next chapter I will map onto Google Web Toolset specific widgets.

In order to improve the ease of use I will be creating one UI which hosts all of the tools. It should also allow the user to use the tools together on a single program in an easy way. A user of the command line toolset can link actions together using UNIX pipes. This, obviously, will not work when using a web frontend so the tools must be combined in another way.

There are a couple ways to combine the tools in one frontend. A simple approach (from design perspective) is just putting all the tools in one screen. This design allows the tools to be used at once in the same screen. The downside of this design is that it does not look attractive. There will also be too much information on the screen at the same time which will make it look cluttered and may lead to confusion. Also, the output of the simulator can become quite large quite fast. It will take over a lot of the screen space and introduce unnecessary scroll bars. Expandability of the user interface will also suffer.

A better design is to use a different view for every tool and tie these views together using a central UI part. The central user interface view contains the most important elements and all other views use this central part to do their job. The user interface consists of different views. These views are designed to cover the functional requirements of each of the tools (listed in previous chapter). Only the view the user has to work with will be displayed. This is the design I used to implement the application.

5.3 VIEWS

In this paragraph the designs of the views are explained. The user interface elements of the views are listed and explained.

5.3.1 PROGRAM BAR

The program bar is the central user interface part which will be visible at all times. The program bar displays the current program to be used by other tools. It also includes the functionality of the parse tool.
1. Program listing
   At the moment a program gets loaded it will appear in the program listing. The program listing lists individual instructions on every line of the list. If the simulator is in use, it will highlight the current instruction.

2. Parse button
   If the parse button is pressed the program loaded in the program list will be parsed.

3. Parse indicator
   The parse indicator indicates using an icon if the program in the program list has proper syntax.

4. Programming language indicator
   Shows the name of the programming language.

5. Basic instruction set indicator
   Shows the basic instruction set used.

5.3.2 SIMULATOR VIEW

The simulator view contains most of the functionality of the simulator tool.
Figure 5: Simulator view

1. Graphical representation.
   Display an image/graph of the current program state.
2. Step button
   Execute a single instruction.
3. Undo button
   Roll back a single instruction.
4. Reset button
   Reset the program counter.

5.3.3 EDITOR VIEW

The editor view allows the user to create a program by entering it into a text field.

Figure 6: Editor view

1. Text field
   A text field which can be edited and holds the users program.
2. **Program list to editor button.**
   If this button is pressed the program currently stored in the program list gets copied into the text field.

3. **Editor to program list button**
   Does the exact opposite of the program list to editor button.

---

### 5.3.4 PROJECTIONS VIEW

The projections view shows the available projections of the current program and can apply these projections.

![Projections View](image)

*Figure 7: Projections view*

1. **Projections list**
   Lists all available projections of the current program and enables the user to pick one.

2. **Project button**
   Executes the projection chosen by the user from the projections list and saves the result in the program list.
6. BUILDING THE USER INTERFACE

It is time to create the user interface. The rough design is made in the previous chapter and now that design has to be mapped on the user interface elements of the Google Web Toolkit. Google Web Toolkit uses so-called ‘widgets’ and panels to implement a user interface. Widgets are user interface elements like buttons, text fields and radio buttons. Panels are the canvas on which widgets (and even other panels) can be placed.

6.1 SELECTING GWT WIDGETS TO IMPLEMENT THE UI ELEMENTS

The application does not need complex user interface elements; it only needs the basic elements like buttons, text fields and drop boxes. Looking at the previous chapter the following user interface elements have to be covered.

- **Button**
  Buttons like the parse button (number 2 of figure 4) are implemented by the Google Web Toolkit as regular HTML buttons.

- **Text field**
  The text field is used by the editor (number 4 of figure 6) and should allow the user to give input. Google Web Toolkit has a widget which implements a standard HTML text field which will suffice.

- **Drop box**
  The drop box which is used by the projection view (number 1 of figure 7) also has an implementation by Google Web Toolkit.

- **Display of an image**
  The display of an image like the parse indicator (number 3 of figure 4) can be done by inserting HTML image tags.

- **Text display**
  Again pure HTML code is used to display text (number 4 of figure 4).

The text field, button and drop box have corresponding widgets in the Google Web Toolkit library. These widgets look like the standard HTML user interface element so the application is recognizable and easy to use. Google Web Toolkit allows these user interface elements to be updated without a browser refresh. It also adds a few methods which make it easy to access these user interface elements from the code. For example text can be written to or read from a text field by the invocation of there methods.

6.2 SELECTING GWT PANELS TO LAYOUT THE UI ELEMENTS

There are a few panels which are useful to the application. I will briefly describe them and explain where and why they are used.

- **Horizontal panel**
  The horizontal panel aligns all embedded widgets or panels horizontally.

- **Vertical panel**
  Same as the horizontal panel but aligns the widgets vertically.

- **Tab panel**
  The tab panel is a panel with one or multiple tabs. If a user clicks on a tab the panel or widget which is assigned to that tab is displayed. Only one panel or widget is shown at a time, the rest is hidden.
The widgets to use were determined in last paragraph, now it is time to place them in panels. It is easier to explain the placement of panels using pictures. Each of the following pictures has different colored boxes which indicate the panels used. The red border indicates a vertical panel is used, the blue border indicates a horizontal panel.

![Diagram](image)

*Figure 8: The use of vertical and horizontal panels in each of the views.*

Only one problem remains; how are the tools going to fit together in the final application? To solve this problem I will use the tab panel mentioned before. Also mentioned before, the program bar will be the main view which is always displayed on the screen. By clicking the tabs the user can choose which view can be seen. Note this tabbed interface provides great expandability; new view can be added to the user interface easily.
Figure 9: The final design of the user interface.
7. BACK-END

In this paragraph the backend of the application is discussed. First I discuss the way Google Web Toolkit handles the much needed remote procedure calls. Next the remote procedure calls themselves are discussed.

7.1 GOOGLE WEB TOOLKIT REMOTE PROCEDURE CALL

The Google Web Toolkit makes making a remote procedure call easy by providing a high level of abstraction. Sending objects and invoking server side code (services) are done in an easy way. Google Web Toolkit also automatically handles serialization of objects. Each service has a small family of helper interfaces or classes, some of which automatically are generated. In order to create a remote procedure call the developer must:

1. Define a synchronous interface for the service.
2. Define a class (according to the synchronous interface) which implements the service.
3. Define an asynchronous interface to the server to be called from the client-side code.

It is not possible for the client side to invoke the service by using the synchronous interface. The nature of asynchronous method calls requires the caller to pass in a callback object that can be notified when an asynchronous call completes, since by definition the caller cannot be blocked until the call completes.

At the client side a callback object has to be defined in which the service can put its return values, if present. Because the remote procedure call can only fail or succeed two methods must also be defined for the callback object which contains code which is executed on a successfully returned call and code which is executed on a failed call.

So by defining the right classes and interfaces remote procedure calls can be done simply by invoking a method on the client side, objects can be sent as arguments of a method and return values can be received using the callback object.

The following piece of code gives an example of how a developer can issue a remote procedure call. We assume here the interfaces ServiceAsync and Service, respectively the asynchronous and synchronous interface, are implemented. Also server side functionality (to be reached using the serviceMethod() method) to handle the remote procedure call is implemented. The focus of this example is the client side of a remote procedure call. The implementation of a remote procedure call of the server site is nothing out of the ordinary. The developer has to implement the core functionality of the service; he/she does not have to handle communication.

```java
// Create the client proxy. Cast the synchronous interface to the asynchronous
// interface. Cast is safe because the generated proxy implements the asynchronous
// interface automatically.
ServiceAsync service = (ServiceAsync)GWT.create(Service.class);

// Create an asynchronous callback to handle the result.
AsyncCallback callback = new AsyncCallback() {
    public void onSuccess(Object result) {
        // This method is invoked on success. 'result' stores the result.
    }

    public void onFailure(Throwable caught) {
```
7.2 REMOTE PROCEDURE CALLS

The remote procedure calls help provide the actual functionality; basically, the user of the web application invokes the PGA toolset using remote procedure calls through the server. And because the PGA toolset is command line based, the web application and the server have to make sure the right command line arguments get passed on the right tools in the PGA toolkit.

The problem is bringing the functionality of the PGA toolset to an online application. Up till now the interfaces between the user and user interface, and user interface and server have been discussed. Now it is time to discuss the interface between the server and the PGA toolset. After some thinking about this interface and its implementation it appeared it could have two forms: stateless or stateful. For each tool in the PGA a decision has to be made; will it get a stateless or stateful interface?

7.2.1 STATELESS VERSUS STATEFUL

I will explain the difference between stateless and stateful using an example. A user wants to upload files to a fileserver. The server is password protected so the user first has to log in to be able to upload files. If the file server is stateful it will function like this:

1. User submits user details (username and password) logs in to the server.
2. User uploads files.
3. User logs out.

The user can upload files as long as he/she is logged in. The server has to keep the session alive as long as the user is logged in.

If the server is stateless it does not have session management. The user will have to supply his/her username and password every time a file is uploaded.

Generally the stateful approach is most efficient (for the server). In case of the example bandwidth is saved because the user does not have to supply his/her credentials everytime he/she wants to upload a file. The main advantage of the stateless approach is its simplicity; the server does not have session management and will, for example, not be concerned with login timeouts.

In the next paragraphs the functionality the PGA toolkit has to provide is inspected. If possible, both a stateful and stateless option to implement the interface is given.

7.2.2 PROGRAM BAR

Parsing is the only functionality (provided by the PGA toolset) the program bar has. The remote procedure call returns the outcome of the parser tool. The parsing tool needs the program itself, the program language and basic instruction set used.
Naturally this interface will be stateless; using sessions will not lead to less bandwidth.

### 7.2.3 PROJECTION VIEW

The projection view has two tasks:

1. **List available projections**
   
   The toolset contains a tool called project which can list which projections have to be made to project one program language onto another. This tool needs the program language where to project from and the program language to project to. This tool can be used to list all available projections of a program if the language to project to is the lowest level language (PGLA). In this case the program outputs all projections which have to be done to reach the lowest level program language. All languages involved in this output can be reached using projections and make up the list of available projections.

2. **Project**
   
   Projecting one language to another can involve multiple projections. All projections needed can be determined using the project tool. Then it will be a matter of combining the individual tools which can project a language to another language.

Both these tasks will be stateless. Again, using sessions will not lead to less bandwidth.

### 7.2.4 SIMULATOR VIEW

The simulator view will contain more complicated functionality than the other views, which are very straightforward. First the functionality is discussed followed by the design of the stateless and stateful method.

- **Step**
  Every time a step is taken in the simulator the visual representation of the current state of the program must be updated.

- **Reset**
  The application is reset in its initial state.

- **Undo**
  Undo the execution of the last instruction.

The current online simulator is stateless; for every step the user makes in the simulated program the whole simulation is run up to the current instruction of the simulation. So with every interaction of the user in the program the execution time of the remote procedure call can increase.

Using the stateful approach the interface will be as follows:

1. When the user loads a program the user gets an individual running copy of the simulator tool on the server.
2. The user can use his/her running copy of the simulator the simulator by issuing step, reset or undo commands. If, for example, the user uses the step command only one instruction is executed on the server in contrast to the stateless method which will execute the whole program again.
3. User quits the application.

The particular advantage of using the stateful method is server side efficiency; instead of executing a whole program just an execution can be performed. The main disadvantage is the
introduction of the need of session management. For example, individual running copies or sudden client disconnects will have to be managed.

For the sake of simplicity I chose the new simulator to be stateless. The inefficiency of the online simulator has never been a problem because of the low number of users. The performance (from a user's perspective) is acceptable. The stateless interface can be rewritten to a stateful interface. The user experience (functionality-wise) will be exactly the same either way.
8. CONCLUSION

This paper has described how the web based application came into existence. It discussed the design of the interface between the user and the toolkit and the interfaces in between.

As for the languages and techniques used, the Google Web Toolkit provided excellent means to create a user interface. It also provided a convenient way to program the interface between the user interface and the server: the remote procedure calls. The Google Web Toolkit was designed for creating applications like the one described in this paper. There are several reasons the Google Web Toolkit became the technique of choice. The Google Web Toolkit yields better performance over the other contestants and provides great debugging options by allowing the developer to run the application in hosted mode. It also had an implementation for asynchronous communication which the others did not have.

The user interface is designed using the functional requirements of each of the tools which will have a web based front end. The examination of these requirements has lead to the addition of an editor with which users can load, save and modify their programs.

The user interface design with the central view and other views providing the functionality of the toolset is in this case the best choice; it hides information which the user is not using, avoiding an overload of information and the tabbed interface provides great expandability for additional views.

We have seen how the Google Web Toolkit makes it possible for the developer to easily implement asynchronous communication between the client side and the server side.

Also the interface between the server and the toolkit was discussed together with the differences, advantages and disadvantages of stateful and stateless implementations of this interface. Stateful methods imply session management which can be tricky to implement. Stateless methods on the other hand are generally less efficient but are easier to implement.

The usage of the Google Web Toolkit has caused the new web based application to be better than the old one. From a user's perspective the new web based application looks better, it is more responsive and because the Google Web Toolkit automatically provides cross browser compatibility the user has a wider choice of web browsers which he/she can use to access the application. It also features more functionality, like projecting or editing, not present in the old web based application.
9. REFERENCES


